RECURSION THEORY

XII

19 January 2022

Gödel's Incompleteness Theorem:

$$\Sigma_1 \neq \Pi_1$$

More precisely: • the set of provable statements (Prov) is Σ_1

• the set of true statements (True) is Π_1-hard

$$\Longrightarrow \mathrm{Prov} \neq \mathrm{True}.$$

We are working in $HF = \bigcup_{n \in \mathbb{N}} V_n$ and we consider $(HF, \in)$ as a model of (finite) set theory.

This requires encoding sets of statements in HF.

<u>Set Theory</u> FST "finite set theory"

HFST FST + only nat. numbers are ordinals

$$(HF, \in) \models HFST$$

Augmented language LST^+:

$\in$, c_x for each $x \in HF$

AHFST is the LST^+-theory HFST + the axioms describing that c_x is the hereditarily finite set x.

$(HF, \in, \{x ; x \in HF\}) \models$ AHFST.

As usual, the first-order theory AHFST cannot uniquely describe the structure HF.
(categorically)

If Σ is an alphabet, identify Σ with an element of HF.
(finite)

So "w is a word" is expressible in LST^+, viz.

"w is a function with $\mathrm{dom}(w)$ is a natural number and $\mathrm{ran}(w) \subseteq c_\Sigma$".

Formulas in LST$^+$ can be identified with "computable" words over Σ:

$$\varphi \longmapsto h(\varphi) \in \Sigma^* \quad \text{bijection}$$
$$\varphi_w \longleftarrow w$$

This allows us to express syntactic statements about formulas in the language of set theory:

E.g.) "w is [the code of] a formula whose first symbol is '$\forall$'."

or "w is [the code of] a formula which has only the variable x as free variable".

Reconstruction of PROVABILITY in HF

Recap our definitions from MLML:

We defined a so called sequent calculus (Sequenzenkalkül). A sequent is just a nonempty sequence of formulas:

$$S = (\underbrace{\varphi_1 \cdots \varphi_n}_{\text{ANTECEDENT}} \underbrace{\underline{\varphi}}_{\text{SUCCEDENT}})$$

It is possible to have $n=0$. Then the antecedent is empty.

S is interpreted as

"$\varphi_1 \wedge \dots \wedge \varphi_n$ imply φ".

As with formulas, we identify sequents with words over Σ as follows:

$$\begin{array}{ccc} S & \longmapsto & h^*(S) \in \Sigma^* \\ S_w & \longleftarrow\!\!\!\shortmid & w \end{array}$$

"computable" bijection

Via this identification, we consider

$$Seq := \{S;\ S \text{ is a sequent}\}$$

as subset of HF:

$$Seq \subseteq HF.$$

This means if g is some syntactic transformation on sequents

$$\begin{tikzcd} w \arrow[r, mapsto] \arrow[d, dashed, "\Sigma^*"'] & S_w \arrow[d, mapsto, "Seq"] \\ h^*(g(S_w)) \arrow[r, leftrightarrow] & g(S_w) \end{tikzcd}$$

is computable.

An n-ary rule is just a subset

$$R \subseteq \mathrm{Seq}^{n+1},$$

interpreted as

"if $(S_1, \dots, S_n, S) \in R$ and I have derived $S_1, \dots, S_n$, then I can derive S".

Rules are written as

$$\frac{S_1 \quad \dots \quad S_n}{S}$$

A calculus is a finite set of rules.

$\mathcal{R}$

Then a sequence $D \subseteq \mathrm{Seq}^*$ is called an $\mathcal{R}$-derivation if for each $i < |D|$ there is an n and an n-ary rule $R \in \mathcal{R}$ and $i_1, \dots, i_n < i$ s.t.

$$(D_{i_1}, \dots, D_{i_n}, D_i) \in R.$$

Once more, we encode elements of Seq^* in HF as usual:

$$\mathcal{D} \longmapsto h^{**}(\mathcal{D}) \in \Sigma^*$$
$$\mathcal{D}_w \longleftarrow\!\shortmid\; w$$

"computable" bijection

So the sets of formulas, sequents and finite sequences of sequents are all identified with Σ^*.

Def. ① $S \in Seq$ is $\mathcal{R}$-derivable if there is an $\mathcal{R}$-derivation $\mathcal{D}$ s.t. S occurs in $\mathcal{D}$.

② If T is a set of formulas, and φ is a formula then

$T \vdash_{\mathcal{R}} \varphi$

if there is an $\mathcal{R}$-derivation $\mathcal{D}$ and a sequent S occurring in $\mathcal{D}$ s.t. all fmlas in the antecedent of S are in T and the succedent of S is φ.

Summary We can write down a formula that describes

$$T \vdash_{\mathcal{Q}} \varphi$$

$$\Longleftrightarrow$$

$\exists w \, \exists v$ D_w is an $\mathcal{Q}$-derivation
and
S_v occurs in D_w
and
all formulas in the antecedent of S_v are in T
and
the succedent of S_v is φ.

Using the operations
$w \longmapsto (w)_0$
$w \longmapsto (w)_1$
with
$(w)_0 * (w)_1 = w$

$$\Longleftrightarrow$$

$\exists w$ $D_{(w)_0}$ is an $\mathcal{Q}$-derivation
and
$S_{(w)_1}$ occurs in $D_{(w)_0}$
and
all formulas in the antecedent of $S_{(w)_1}$ are in T
and
the succedent of $S_{(w)_1}$ is φ.

Looks like a Σ_1 description.

Remark. $T \subseteq \Sigma^*$ by identification h
$R \in \mathcal{R}$
$R \subseteq (\Sigma^*)^{n+1}$ by identification h^*
So if T, R are not computable, there is no reason for the above description to be really Σ_1.

Definition ① A rule $R \subseteq Seq^{n+1}$ is called computable if the set

$$\{\underbrace{(\cdots(w_1 * w_2) * w_3 * \cdots) * w)}_{n+1} \,;\, (S_{w_1}, \dots, S_{w_n}, S_w) \in R\}$$

$$\subseteq \Sigma^*$$

is computable.
[So, intuitively, R is computable if checking whether a given tuple satisfies it is computable.]

② A calculus $\mathcal{R}$ is called computable if all of its rules are computable.

Remark Since we identify formulas with words, it's already clear what it means for T to be computable.

Theorem If $\mathfrak{R}$ is a computable calculus and T is a computable set of formulas, then

$$\underline{\underline{\mathrm{Prov}_{\mathfrak{R}}}} := \{ \underline{w} ; T \vdash_{\mathfrak{R}} \varphi_w \}$$

is computably enumerable (Σ_1).

Proof. Remember the earlier formula

$T \vdash_{\mathfrak{R}} \varphi_w$

$\Longleftrightarrow \exists v$ [(a) $D_{(v)_0}$ is an $\mathfrak{R}$-derivation and
(b) $S_{(v)_1}$ occurs in $D_{(v)_0}$ and
(c) the succedent of $S_{(v)_1}$ is φ_w and
(d) all formulas in the antecedent of $S_{(v)_1}$ are in T]

$A(v,w)$

So, the question is: given v and w, can I compute whether the four conditions (a)–(d) are satisfied?

Since being computable is closed under intersections, it's enough to check each of the conditions (a) – (d) separately.

(a) $\mathcal{D}_{(v)_0}$ is an $\mathcal{R}$-derivation

$$v \longmapsto (v)_0 \quad \text{[computable since } (\cdot)_0 \text{ is]}$$
$$\downarrow$$
$$\mathcal{D}_{(v_0)}$$

Now find $N := |\mathcal{D}_{(v)_0}|$ and check for each $i < N$ whether the condition of being an $\mathcal{R}$-derivation is satisfied:
So for each i go through the finitely many rules $R \in \mathcal{R}$ and [if R is n-ary] check every single tuple $(i_1, \dots, i_n)$ with $i_1, \dots, i_n < i$ whether

$$\left((\mathcal{D}_{(v)_0})_{i_1}, \dots, (\mathcal{D}_{(v)_0})_{i_n}, (\mathcal{D}_{(v)_0})_i\right) \in R$$

For each i, this requires finitely many checks and each of them is computable. Furthermore, there are only finitely many is.

(b) $S_{(v)_1}$ occurs in $D_{(v)_0}$

*

(b) & (c) do not require the assumptions of the theorem

Similar: $v \longmapsto (v)_0 \longmapsto D_{(v)_0}$

$ \longmapsto (v_1) \longmapsto S_{(v)_1}$.

Check (up to $N := |D_{(v)_0}|$) whether one of the elements of $D_{(v)_0}$ is equal to $S_{(v)_1}$.

(c) the succedent of $S_{(v)_1}$ is φ_w.

*

Similar: $v \longmapsto (v)_1 \longmapsto S_{(v)_1}$.

This is a sequence of formulas; identify the last one and check the last one is equal to φ_w.

(d) all formulas in the antecedent of $S_{(v)_1}$ are in T.

<u>Similar</u>: $v \longmapsto (v)_1 \longmapsto S_{(v)_1}$

Identify all formulas in the antecedent this is n if $|S_{(v)_1}| = n+1$.

Since T is computable, these are n computable checks.

This shows that

$$\{v*w \,;\, A(v,w)\} = A^*$$

is computable and thus

$$\mathrm{Prov}_{\mathcal{R}} = \{w \,;\, \exists v \; v*w \in A^*\}$$

is Σ_1, thus c.e.

q.e.d.

<u>Corollary</u> If $\mathcal{R}_G$ is the standard Gentzen calculus for first-order logic, as defined in MLML, and T is computable, then

$$\mathrm{Prov} := \{w \,;\, T \vdash_{\mathcal{R}_G} \varphi_w\}$$

is Σ_1.

<u>Proof.</u> We only need to show that $\mathcal{R}_G$ is computable, i.e., each $R \in \mathcal{R}_G$ is computable.

Gentzen's Sequent Calculus

For the reader's convenience, we list all the rules of $\mathfrak{S}$ together.

(Assm) $\dfrac{}{\Gamma \quad \varphi}$ if $\varphi \in \Gamma$

(Ant) $\dfrac{\Gamma \quad \varphi}{\Gamma' \quad \varphi}$ if $\Gamma \subset \Gamma'$

(PC) $\dfrac{\begin{array}{lll}\Gamma & \psi & \varphi \\ \Gamma & \neg\psi & \varphi\end{array}}{\begin{array}{lll}\Gamma & & \varphi\end{array}}$

(Ctr) $\dfrac{\begin{array}{lll}\Gamma & \neg\varphi & \psi \\ \Gamma & \neg\varphi & \neg\psi\end{array}}{\begin{array}{lll}\Gamma & & \varphi\end{array}}$

(∨A) $\dfrac{\begin{array}{lll}\Gamma & \varphi & \chi \\ \Gamma & \psi & \chi\end{array}}{\begin{array}{lll}\Gamma & (\varphi \vee \psi) & \chi\end{array}}$

(∨S) $\dfrac{\Gamma \quad \varphi}{\Gamma \quad (\varphi \vee \psi)}, \quad \dfrac{\Gamma \quad \varphi}{\Gamma \quad (\psi \vee \varphi)}$

(∃A) $\dfrac{\Gamma \quad \varphi\frac{y}{x} \quad \psi}{\Gamma \quad \exists x\varphi \quad \psi}$ if y is not free in $\Gamma\ \exists x\varphi\ \psi$

(∃S) $\dfrac{\Gamma \quad \varphi\frac{t}{x}}{\Gamma \quad \exists x\varphi}$

(≡) $\dfrac{}{t \equiv t}$

(Sub) $\dfrac{\Gamma \quad \varphi\frac{t}{x}}{\Gamma \quad t \equiv t' \quad \varphi\frac{t'}{x}}$

Ebbinghaus, Flum, Thomas (English version), IV.6, p. 69

Check that (PC) is computable:

(PC) is a 2-ary rule, $(PC) \subseteq Seq^{2+1} = Seq^3$

Let (S_1, S_2, S) be an arbitrary elt of Seq^3

The following is checkable by a computer:

S_1 has length $k+2$, S_2 has length $k+2$, S has length $k+1$.

The first k formulas of S_1, S_2, S are the same.

The $k+2$nd fmla of S_1, S_2 and $k+1$st fmla of S are the same.

The $k+1$st fmla of S_2 is $\neg\psi$ where ψ is the $k+1$st fmla S_1

This (tedious) check shows that Q_G is computable, so the corollary follows.

q.e.d.

Note that this result does not rely on a specific theory (such as PA, FST, HFST, AHFST), but is completely general:

> As long as the set of axioms of a theory T is computable, the set of T-provable formulas is c.e.

Reconstruction of computability in LST$^+$

Going back to the beginning of the course, we remember how we defined computability:

$$\mathbb{M} = (\Sigma, \Phi, h) \quad \text{model of computation}$$

$$P \in \Sigma^* \text{ program}; \quad w \in \Sigma^* \text{ input}$$

Based on P, w, we defined the M-computation of P with input w.

$$C_M(P, w, i) \quad \text{where } i \in \mathbb{N}$$

Each time instance ($i \in \mathbb{N}$) is a configuration of the machine.

We said that the computation halts, if there is $i \in \mathbb{N}$ s.t.

$$C_M(P, w, i) \text{ is in the halting state.}$$

This means that the statement

(*) "the M-computation of P with input w halts"

is true iff there is a finite sequence $C_0, \dots, C_n$ s.t. each C_i is a configuration

$$C_i = C_M(P, w, i)$$

and C_n is in the halting state.

If $P, w \in HF$, then there is some formula $\varphi \in LST^+$ s.t. $\varphi(P, w) \Longleftrightarrow (*)$.